

Fully Dynamic Data Structure for Top- k Queries on Uncertain Data

Manish Patil ¹, Rahul Shah ², Sharma V. Thankachan ³

Computer Science Department, Louisiana State University,
Baton Rouge, LA, USA

¹mpatil@csc.lsu.edu

²rahul@csc.lsu.edu

³thanks@csc.lsu.edu

Abstract—Top- k queries allow end-users to focus on the most important (top- k) answers amongst those which satisfy the query. In traditional databases, a user defined score function assigns a score value to each tuple and a top- k query returns k tuples with the highest score. In uncertain database, top- k answer depends not only on the scores but also on the membership probabilities of tuples. Several top- k definitions covering different aspects of score-probability interplay have been proposed in recent past [10], [4], [2], [8]. Most of the existing work in this research field is focused on developing efficient algorithms for answering top- k queries on static uncertain data. Any change (insertion, deletion of a tuple or change in membership probability, score of a tuple) in underlying data forces re-computation of query answers. Such re-computations are not practical considering the dynamic nature of data in many applications. In this paper, we propose a fully dynamic data structure that uses ranking function $PRF^e(\alpha)$ proposed by Li et al. [8] under the generally adopted model of x -relations [11]. PRF^e can effectively approximate various other top- k definitions on uncertain data based on the value of parameter α . An x -relation consists of a number of x -tuples, where x -tuple is a set of mutually exclusive tuples (up to a constant number) called alternatives. Each x -tuple in a relation randomly instantiates into one tuple from its alternatives. For an uncertain relation with N tuples, our structure can answer top- k queries in $O(k \log N)$ time, handles an update in $O(\log N)$ time and takes $O(N)$ space. Finally, we evaluate practical efficiency of our structure on both synthetic and real data.

Index Terms—ignore

I. INTRODUCTION

The efficient processing of uncertain data is an important issue in many application domains because of the imprecise nature of data they generate. The nature of uncertainty in data is quite varied, and often depends on the application domain. In response to this need, much effort has been devoted to modeling uncertain data [11], [3], [1], [7], [9]. Most models have been adopted to possible world semantics, where an uncertain relation is viewed as a set of possible instances (worlds) and correlation among the tuples governs generation of these worlds.

Consider traffic monitoring application data [10] (with modified probabilities) as shown in Table I, where radar is used to detect car speeds. In this application, data is inherently uncertain because of errors in reading introduced by nearby high voltage lines, interference from near by car, human operator error etc. If two radars at different locations detect the

TABLE I
TRAFFIC MONITORING DATA: $t_1, \{t_2, t_4\}, \{t_3, t_6\}, t_5$

Time	Car Loc	Plate No	Speed	Prob	Tuple Id
11:55	L1	Y-245	130	0.30	t_1
11:40	L2	X-123	120	0.40	t_2
12:05	L3	Z-541	110	0.20	t_3
12:15	L4	X-123	105	0.50	t_4
12:10	L5	L-110	95	0.30	t_5
11:35	L6	Z-541	80	0.45	t_6

presence of the same car within a short time interval, such as tuples t_2 and t_4 as well as t_3 and t_6 , then at most one radar reading can be correct. We use x -relation model to capture such corrections. An x -tuple τ specifies a set of exclusive tuples, subject to the constraint $\sum_{t_i \in \tau} Pr(t_i) \leq 1$. The fact that t_2 and t_4 cannot be true at the same time, is captured by the x -tuple $\tau_1 = \{t_2, t_4\}$. Similarly $\tau_2 = \{t_3, t_6\}$. Probability of a possible world is computed based on the existence probabilities of tuples present in a world and absence probabilities of tuples in the database that are not part of a possible world. For example, consider the possible world $pw = \{t_1, t_2, t_3\}$. Its probability is computed by assuming the existence of t_1, t_2, t_3 , and the absence of t_4, t_5 , and t_6 . However since t_2 and t_4 are mutually exclusive presence of tuple t_2 implies absence of t_4 and same is applicable for tuples t_3 and t_6 . Therefore, $Pr(pw) = 0.3 \times 0.4 \times 0.2 \times (1 - 0.3) = 0.0168$.

Top- k queries on a traditional certain database have been well studied. For such cases, each tuple is associated with a single score value assigned to it by a scoring function. There is a clear total ordering among tuples based on score, from which the top- k tuples can be retrieved. However, for answering a top- k query on uncertain data, we have to take into account both, ordering based on scores and ordering based on existence probabilities of tuples. Depending on how these two orderings are combined, various top- k definitions with different semantics have been proposed in recent times. Most of the existing work studies only the problem of answering a top- k query on a static uncertain data. Though the query time of an algorithm depends on the choice of a top- k definition, linear scan of tuples achieves the best bound so far. Therefore, recomputing top- k answers in

an application with frequent insertions and deletions can be extremely inefficient. In this paper, we present a fully dynamic structure of size $O(N)$ that always maintains the correct answer to the top- k query for an uncertain database. The structure is based on a decomposition of the problem so that updates can be handled efficiently. Our structure can answer the top- k query in $O(k \log N)$ time, handle update in $O(\log N)$ time.

Outline: In Section II we review different top- k definitions proposed so far and try to compare them against a parameterized ranking function $PRF^e(\alpha)$ proposed by Li et al. [8]. We choose $PRF^e(\alpha)$ over other definitions as it can approximate many of the other top- k definitions and can handle data updates efficiently. After formally defining the problem (Section III), we explain how $PRF^e(\alpha)$ can be computed using divide and conquer approach (Section IV), which forms the basis of our data structure explained in Section V. We present experimental study with real and synthetic data sets in Section VI. Finally we review the related work in Section VII before concluding the paper.

II. TOP- k QUERIES ON UNCERTAIN DATA

Soliman et al. [10] first considered the problem of ranking tuples when there is both a score and probability for each tuple. Several other definitions of ranking have been proposed since then for probabilistic data.

- Uncertain Top- k (U-Topk) [10]: It returns a k -tuple set that appears as top- k answer in possible worlds with maximum probability.
- Uncertain Rank- k (U-Ranks) [10]: It returns a tuple for each i , such that it has maximum probability of appearing at rank i across all possible worlds.
- Probabilistic Threshold Query (PT- k) [4]: It returns all the tuples with probability of appearing in top- k greater than a user specified threshold.
- Expected Rank (E-Rank) [2]: k tuples with highest value of expected rank ($er(t_i)$) are returned.

$$er(t_i) = \sum Pr(pw)rank_{pw}(t_i)$$

where $rank_{pw}(t_i)$ denotes rank of t_i in a possible world pw . In case t_i does not appear in possible world, $rank_{pw}(t_i)$ is defined as $|pw|$.

- Expected Score (E-Score) [2]: k tuples with highest value of expected score ($es(t_i)$) are returned.

$$es(t_i) = Pr(t_i)score(t_i)$$

- Parameterized Ranking Function (PRF) [8]: PRF in its most general form is defined as,

$$\Upsilon(t_i) = \sum_r w(t_i, r) \times Pr(t_i, r) \quad (1)$$

where w is the weight function that maps a given tuple-rank pair to a complex number and $Pr(t_i, r)$ denotes

the probability of a tuple t_i being ranked at position r across all possible worlds. A top- k query returns those k tuples with the highest Υ values. Different weight functions can be plugged in to the above definition to get a range of ranking functions, subsuming most of top- k definitions listed above. A special ranking function $PRF^e(\alpha)$ is obtained by choosing $w(t_i, r) = \alpha^{r-1}$, where α is a constant. Experimental study in [8] reveals that for some value of α with the constraint $\alpha < 1$, PRF^e can approximate many existing top- k definitions.

Algorithms for computing top- k answers using the above ranking functions have been studied for static data. Any changes in the underlying data forces re-computation of query answers. To understand the impact of a change on top- k answers, we analyze relative ordering of the tuples before and after a change, based on these ranking functions.

Let $T = t_1, t_2, \dots, t_N$ denote independent tuples sorted in non-increasing order of their score. We choose insertion of a tuple as a representative case for changes in T , and monitor its impact on relative ordering of a pair of tuples (t_i, t_j) . Since E-Score of a tuple depends only on its score and existence probability, ordering is preserved for all (t_i, t_j) pairs in T . For ranking functions U-Ranks, PT- k ordering of tuples (t_i, t_j) may or may not be preserved by insertion and cannot be guaranteed when the score of a new tuple is higher than that of t_i and t_j . Hence, existing top- k answers do not provide any useful information for re-computation of query answers. E-Rank further complicates the matter as expected rank of a tuple depends on both higher and lower scored tuples. However, when tuples are ranked using $PRF^e(\alpha)$, the scope of disturbance in the relative ordering of tuples is limited as explained in later sections. This enables efficient handling of updates in the database. Therefore, this ranking function is well suited for answering top- k queries on a dynamic collection of tuples.

III. PROBLEM STATEMENT

Given an uncertain relation T of a dynamic collection of tuples, such that each tuple $t_i \in T$ is associated with a membership probability value $Pr(t_i) > 0$ and a score $score(t_i)$ computed based on a scoring function, the goal is to retrieve the Top- k tuples.

We use the parameterized ranking function $PRF^e(\alpha)$ proposed by [8] in this paper. $PRF^e(\alpha)$ is defined as,

$$\Upsilon(t_i) = \sum_r \alpha^{r-1} \times Pr(t_i, r) \quad (2)$$

where α is a constant and $Pr(t_i, r)$ denotes the probability of a tuple t_i being ranked at position r across all possible worlds*. A top- k query returns the k tuples with highest Υ values. We refer to $\Upsilon(t_i)$ as the rank-score of tuple t_i . In this paper, we adopt the x -relation model to capture corrections. An x -tuple τ specifies a set of exclusive tuples,

* $Pr(t_i, r) = 0$, for $r > i$.

subject to the constraint $\sum_{t_i \in \tau} Pr(t_i) \leq 1$. In a randomly instantiated world τ takes t_i with probability $Pr(t_i)$, for $i = 1, 2, \dots, |\tau|$ or does not appear at all with probability $1 - \sum_{t_i \in \tau} Pr(t_i)$. Here $|\tau|$ represents the number of tuples belonging to set τ . Let $\tau(t_i)$ represents an x -tuple to which tuple t_i belongs to. In x -relation model, T can be thought of as a collection of pairwise-disjoint x -tuples. Let $\sum_{\tau \in T} |\tau| = N$ i.e. there are total N tuples in an uncertain relation T . Without loss of generality, we assume all scores to be unique and let t_1, t_2, \dots, t_N denotes ordering of the tuples in T when sorted in descending order of the score ($score(t_i) > score(t_{i+1})$). From now onwards we represent $Pr(t_i)$ by short notation p_i for simplicity.

IV. COMPUTING $PRF^e(\alpha)$

In this section, we derive a closed form expression for the rank-score $\Upsilon(t_i)$, followed by an algorithm for retrieving the Top-1 tuple from a collection of independent tuples. In the next section we show that this approach can be easily extended to a data structure for efficiently retrieving Top- k tuples from a dynamic collection of tuples. We begin by assuming tuple independence and then consider correlated tuples, where correlations are represented using x -tuples.

A. Assuming tuple independence:

When all tuples are independent, tuple t_i appears at position r in a possible world pw if and only if exactly $(r - 1)$ tuples with a higher score value appear in pw . Let $S_{i,r}$ be the probability that a randomly generated world from $\{t_1, t_2, \dots, t_i\}$ has exactly r tuples. Then, probability of a tuple t_i being ranked at r is given as

$$Pr(t_i, r) = p_i S_{i-1, r-1} \quad (3)$$

where,

$$S_{i,r} = \begin{cases} p_i S_{i-1, r-1} + (1 - p_i) S_{i-1, r} & \text{if } i \geq r > 0 \\ 1 & \text{if } i = r = 0 \\ 0 & \text{otherwise.} \end{cases}$$

Using above recursion for $S_{i,r}$ and equation 2, 3,

$$\begin{aligned} \Upsilon(t_i) &= \sum_r \alpha^{r-1} Pr(t_i, r) = \sum_r \alpha^{r-1} p_i S_{i-1, r-1} \\ \frac{\Upsilon(t_i)}{p_i} &= \sum_r \alpha^{r-1} S_{i-1, r-1} = \sum_r \alpha^r S_{i-1, r} \end{aligned}$$

Similarly,

$$\begin{aligned} \frac{\Upsilon(t_{i+1})}{p_{i+1}} &= \sum_r \alpha^r S_{i,r} \\ &= \sum_r \alpha^r (p_i S_{i-1, r-1} + (1 - p_i) S_{i-1, r}) \\ &= \alpha p_i \sum_r \alpha^{r-1} S_{i-1, r-1} + (1 - p_i) \sum_r \alpha^r S_{i-1, r} \\ &= (1 - (1 - \alpha) p_i) \Upsilon(t_i) / p_i \end{aligned}$$

We have the base case, $\Upsilon(t_1) = p_1$. Therefore,

$$\Upsilon(t_i) = p_i \prod_{j < i} (1 - (1 - \alpha) p_j) \quad (4)$$

Now, we analyze the contribution of a tuple t_i towards global ranking over T using the above formula as follows.

- Tuple t_i contributes $m_i = p_i$ for the computation of its own rank-score.
- Tuple t_i contributes $c_i = 1 - (1 - \alpha) p_i$ of computing rank-score for all tuples having score less than that of t_i .

Theorem 1: When all tuples in T are independent, rank-score of a tuple t_i can be computed as follows,

$$\Upsilon(t_i) = m_i \prod_{j < i} c_j$$

where $m_i = p_i$ and $c_j = 1 - (1 - \alpha) p_j$ □

Answering Top-1 query:

We use a divide and conquer approach for answering top-1 query on T , which forms the basis for our data structure in later section. Let the given relation $T = \{t_1, t_2, \dots, t_N\}$ be partitioned into sub-relations $T_l = \{t_1, t_2, \dots, t_{\lceil N/2 \rceil}\}$ and $T_r = \{t_{\lceil N/2 \rceil + 1}, t_{\lceil N/2 \rceil + 2}, \dots, t_N\}$. Also let t^l and t^r represent the top-1 answer for T_l and T_r with rank-scores $\Upsilon_{T_l}(t^l)$ and $\Upsilon_{T_r}(t^r)$ respectively, where $\Upsilon_{T_l}(t^l)$ is computed by considering only those tuples $t_j \in T_l$ and $\Upsilon_{T_r}(t^r)$ is computed by considering only those tuples $t_j \in T_r$.

For $t_i \in T_l$,

$$\Upsilon_{T_l}(t_i) = m_i \prod_{\substack{j < i \\ t_j \in T_l}} c_j$$

and similarly for $t_i \in T_r$,

$$\Upsilon_{T_r}(t_i) = m_i \prod_{\substack{j < i \\ t_j \in T_r}} c_j$$

Now when both relations T_l and T_r are merged to form T , we make the following observations using the above analysis:

- The contribution of each tuple towards its own rank-score remains unchanged.
- Since all the tuples in T_r have a lower score value than any tuple $t_i \in T_l$ they do not contribute towards the rank-score value of t_i computed over entire relation T . Thus $\Upsilon(t_i) = \Upsilon_{T_l}(t_i)$. Hence t^l still has the highest rank-score value $\Upsilon(t^l)$ among the tuples in T_l .
- Since all the tuples in T_l have higher score value than any tuple $t_i \in T_r$, each $t_j \in T_l$ contributes $1 - (1 - \alpha) p_j$ towards rank-score value of t_i computed over entire relation T . Let $C_l = \prod_{t_j \in T_l} c_j = \prod_{t_j \in T_l} 1 - (1 - \alpha) p_j$ represents overall contribution of sub-relation T_l . Then $\Upsilon(t_i) = C_l \Upsilon_{T_r}(t_i)$. Since rank-score value of every tuple $t_i \in T_r$ gets scaled by the same factor C_l , t^r still has

the highest rank-score value $\Upsilon(t^r)$ among the tuples in T_r .

Therefore the top-1 answer over uncertain relation T can be chosen from t^l and t^r based on the their rank-score values computed over the entire relation.

B. Supporting correlations

If t_i has some preceding alternatives, then the event that t_i appears is no longer independent of the event that exactly $j - 1$ tuples appear in $\{t_1, t_2, \dots, t_{i-1}\}$, as in equation 3. Hence equation 4 cannot be used to compute the rank-score of a tuple t_i . To overcome this difficulty, we convert the relation T to \bar{T}^i where all the tuples are independent [12]. Let $\tau^i = \{t_j | t_j \in \tau, j < i\}$. Now for each x -tuple $\tau \in T$, we create an x -tuple $\bar{\tau} = \{\bar{t}\}$ in \bar{T}^i , where $p(\bar{t}) = Pr(\tau^i)$ with one exception. For tuple $\bar{t} \in \bar{T}^i$ which corresponds to $\tau(t_i) \in T$, we use $Pr(\bar{t}) = p_i$, where $\tau(t_i)$ is the x -tuple to which the tuple t_i belongs to.

For example, $T = \{\tau_1, \tau_2, \tau_3\}$ where, $\tau_1 = \{t_1, t_3, t_6\}$, $\tau_2 = \{t_2, t_7\}$ and $\tau_3 = \{t_4, t_5\}$. Then $\tau_1^5 = \{t_1, t_3\}$ and $\tau(t_5) = \tau_3$.

This conversion takes into account the fact that only tuples with a score higher than that of t_i contribute to $Pr(t_i, r)$ as well as to $\Upsilon(t_i)$, and the presence of t_i implies absence of all its related tuples.

Since all the tuples in \bar{T}^i are independent among themselves, we can now use equation 4 on \bar{T}^i to compute the rank-score of tuple t_i . Combining related tuples into a representative tuple \bar{t} does not affect $\Upsilon(t_i)$ here, since the probability that \bar{t} appears is the same as the probability that one tuple in $\tau \in T$ with score higher than $score(t_i)$ appears. Therefore,

$$\begin{aligned} \Upsilon(t_i) &= p_i \prod_{\substack{\bar{t} \in \bar{T}^i \\ \bar{\tau}(\bar{t}) \neq \tau(t_i)}} (1 - (1 - \alpha)Pr(\bar{t})) \\ &= p_i \prod_{\substack{\tau \in T \\ \tau \neq \tau(t_i)}} (1 - (1 - \alpha)Pr(\tau^i)) \end{aligned} \quad (5)$$

Now, we analyze the contribution of an x -tuple towards global ranking over T using the above formula as follows.

- x -tuple τ contributes $m_i = p_i$ for computing rank-score of a tuple $t_i \in \tau$.
- x -tuple τ contributes $c_i = 1 - (1 - \alpha)Pr(\tau^i)$ for computing rank-score of a tuple $t_i \notin \tau$.

Answering Top-1 query:

Again, we attempt to use a divide and conquer algorithm for answering top-1 query on T by partitioning relation $T = \{t_1, t_2, \dots, t_N\}$ into sub-relations $T_l = \{t_1, t_2, \dots, t_{\lceil N/2 \rceil}\}$ and $T_r = \{t_{\lceil N/2 \rceil + 1}, t_{\lceil N/2 \rceil + 2}, \dots, t_N\}$ and assuming t^l and t^r represent the top-1 answers for T_l and T_r respectively. If property that t^l and t^r remains highest rank-score tuples in their respective sub-relations even after merging of T_l and T_r , holds true then reporting top-1 for relation T can be done by simply comparing rank-score values of t^l and t^r over entire

relation T . Unfortunately, this property may not hold true for t^r .

To illustrate the problem, consider an uncertain relation $T = \{t_1, t_2, t_3, t_4\}$ with $p_1 = 0.35, p_2 = 0.3, p_3 = 0.4, p_4 = 0.45$ and tuples t_2 and t_3 are mutually exclusive. Using equation 5, rank-scores can be computed as follows ($\alpha = 0.8$):

$$\begin{aligned} \Upsilon(t_1) &= 0.35 \\ \Upsilon(t_2) &= 0.3(1 - 0.2 \times 0.35) = 0.28 \\ \Upsilon(t_3) &= 0.4(1 - 0.2 \times 0.35) = 0.37 \\ \Upsilon(t_4) &= 0.45(1 - 0.2 \times 0.35)(1 - 0.2 \times (0.3 + 0.4)) = 0.36 \end{aligned}$$

Top-1 query on T should return tuple t_3 with highest rank-score value 0.37. By adopting the divide and conquer approach to tackle the problem, we partition the given relation into $T_l = \{t_1, t_2\}$ and $T_r = \{t_3, t_4\}$. Top-1 query is applied to these sub-relations as follows.

$$\begin{aligned} \Upsilon_{T_l}(t_1) &= 0.35 \\ \Upsilon_{T_l}(t_2) &= 0.3(1 - 0.2 \times 0.35) = 0.28 \end{aligned}$$

$$\begin{aligned} \Upsilon_{T_r}(t_3) &= 0.4 \\ \Upsilon_{T_r}(t_4) &= 0.45(1 - 0.2 \times 0.4) = 0.41 \end{aligned}$$

Thus t_1 and t_4 will be reported from T_l and T_r as top-1 answers respectively. By simple merge operation, which computes rank-score values for t_1, t_4 over relation T and compares them, t_1 will be reported as top-1 answer for T . However actual top-1 answer is tuple t_3 . The fact that dependance of t_2 and t_3 was ignored while answering top-1 over sub-relation T_r is the root cause behind the disturbance in relative ordering of t_3 and t_4 .

Therefore in order to maintain the relative ordering of tuples based on their rank-score over entire relation during merge, we redefine the expressions for contributions as follows. Here we use the notation \hat{p}_i for sum of probabilities of all tuples t_j which are related to t_i and have score greater than the score of t_i (i.e. $j < i$). In the above example $\hat{p}_3 = p_2 = 0.3$.

$$\hat{p}_i = Pr([\tau(t_i)]^i) = \sum_{\substack{\tau(t_i) = \tau(t_j) \\ j < i}} p_j$$

Now equation 5 can be re arranged as follows,

$$\Upsilon(t_i) = \frac{p_i}{(1 - (1 - \alpha)\hat{p}_i)} \prod_{\tau \in T} (1 - (1 - \alpha)Pr(\tau^i))$$

$$\frac{\Upsilon(t_i)}{m_i} = \prod_{\tau \in T} (1 - (1 - \alpha)Pr(\tau^i))$$

$$\text{where } m_i = \frac{p_i}{(1 - (1 - \alpha)\hat{p}_i)}$$

similarly,

$$\frac{\Upsilon(t_{i+1})}{m_{i+1}} = \prod_{\tau \in T} (1 - (1 - \alpha)Pr(\tau^{i+1}))$$

Here note that $Pr(\tau^i) = Pr(\tau^{i+1})$ for all $\tau \neq \tau(t_i)$. From the above two equations,

$$\begin{aligned} \left(\frac{\Upsilon(t_{i+1})}{m_{i+1}} \right) / \left(\frac{\Upsilon(t_i)}{m_i} \right) &= \frac{1 - (1 - \alpha)Pr([\tau(t_i)]^{i+1})}{1 - (1 - \alpha)Pr([\tau(t_i)]^i)} \\ &= \frac{1 - (1 - \alpha)(\hat{p}_i + p_i)}{1 - (1 - \alpha)\hat{p}_i} \\ &= c_i \end{aligned}$$

The base case is $\Upsilon(t_1) = p_1$. Therefore we can rewrite equation 5 as follows,

$$\frac{\Upsilon(t_{i+1})}{m_{i+1}} = c_i \frac{\Upsilon(t_i)}{m_i} = c_i c_{i-1} \frac{\Upsilon(t_{i-1})}{m_{i-1}} = \dots = \prod_{j \leq i} c_j \quad (6)$$

The result is summarized in following theorem.

Theorem 2: For an uncertain relation T , rank-score of a tuple t_i can be computed as,

$$\Upsilon(t_i) = m_i \prod_{j < i} c_j$$

where $m_i = \frac{p_i}{(1 - (1 - \alpha)\hat{p}_i)}$, $c_i = \frac{1 - (1 - \alpha)(\hat{p}_i + p_i)}{1 - (1 - \alpha)\hat{p}_i}$ and $\hat{p}_i = \sum t_r$, where t_i and t_r are mutually exclusive and $r < i$. \square

This equation is applicable for dependent as well as independent tuples. Note that here m_i and c_i are dependent only on the tuples which are related to t_i , hence can be computed/updated efficiently. Moreover, the contribution c_i of a tuple t_i to the rank-score of a tuple t_j is the same for all $j > i$. Hence, the relative ordering will not change even if we use our divide and conquer approach.

Consider the same example as before. We begin by computing values of m_i and c_i for each tuple.

$$\begin{aligned} m_1 &= 0.35 \\ m_2 &= 0.3 \\ m_3 &= 0.4 / (1 - 0.2 \times 0.3) = 0.43 \\ m_4 &= 0.45 \end{aligned}$$

$$\begin{aligned} c_1 &= (1 - 0.2 \times 0.35) = 0.93 \\ c_2 &= (1 - 0.2 \times 0.3) = 0.94 \\ c_3 &= (1 - 0.2 \times (0.3 + 0.4)) / (1 - 0.2 \times 0.3) = 0.91 \\ c_4 &= (1 - 0.2 \times 0.45) = 0.91 \end{aligned}$$

Now, we partition T into $T_l = \{t_1, t_2\}$ and $T_r = \{t_3, t_4\}$ and apply Top-1 query to these sub-relations.

$$\begin{aligned} \Upsilon_{T_l}(t_1) &= m_1 = 0.35 \\ \Upsilon_{T_l}(t_2) &= m_2 \times c_1 = 0.3 \times 0.94 = 0.28 \end{aligned}$$

$$\begin{aligned} \Upsilon_{T_r}(t_3) &= m_3 = 0.43 \\ \Upsilon_{T_r}(t_4) &= m_4 \times c_3 = 0.45 \times 0.91 = 0.41 \end{aligned}$$

It can be seen that from t_1 and t_3 are chosen as Top-1 from T_l and T_r respectively. During next comparison, t_3 ($\Upsilon(t_3) = m_3 \times c_1 \times c_2 = 0.37$) will be reported as the Top-1 tuple, which is correct.

V. OUR DATA STRUCTURE:

In the earlier sections, we derived the simple closed form expression for calculating $\Upsilon(t_i)$ for a tuple t_i . Now our task is to maintain a dynamic collection of tuples, such that for a given query k , we retrieve Top- k rank-scored tuples efficiently. We use data structural approach for this problem. Our structure is a balanced binary search tree Δ such that each leaf corresponds to a tuple in an uncertain relation T . Moreover, leaves in the tree are sorted in decreasing order of the score i.e. leaves $\ell_1, \ell_2, \dots, \ell_N$ of the tree represent tuples t_1, t_2, \dots, t_N in the same order from left to right, such that $score(t_i) > score(t_{i+1})$. Let T_u represents the sub-relation containing tuples associated with leaves of a subtree rooted at node u . i.e. $T_u = \{t_{u'}, t_{u'+1}, \dots, t_{u''}\}$ and $\ell_{u'}$ represents the left-most and $\ell_{u''}$ represents the right-most leaf of node u . At each node u , we store a triplet (top_u, M_u, C_u) such that:

- top_u is the tuple (represented by ℓ_{u^*}) with highest rank-score among tuples in sub-relation T_u . Here $u' \leq u^* \leq u''$.
- M_u is the contribution of all tuples in T_u towards rank-score of tuple top_u .

$$M_u = m_{u^*} \prod_{u' \leq i < u^*} c_i$$

- C_u is the contribution of all tuples in T_u towards tuple t_i such that $i > u''$, where $\ell_{u''}$ is the right-most leaf of the subtree rooted at node u .

$$C_u = \prod_{u' \leq i < u''} c_i$$

Since our data structure stores only a constant number of information at each node, and the number of nodes are bounded by $O(N)$, the total space requirement of our data structure is $O(N)$.

If node u is a leaf node representing the tuple t_i , then $M_u = m_i$, $top_u = t_i$ and $C_u = c_i$. If u is an internal node, this information can be computed using the MERGE operation given below. Figure 1 shows an example for the uncertain data in table II.

MERGE (u)

$$\begin{aligned} v &= left-child(u) \\ w &= right-child(u) \\ M_u &= \max(M_v, C_v \times M_w) \\ top_u &= top_v, \text{ if } M_v > C_v \times M_w, \text{ else } top_u = top_w \\ C_u &= C_w \times C_v \end{aligned}$$

Theorem 3: The data structure Δ maintains a dynamic collections of tuples such that Top-1 tuple, $t^1 = top_{root}$ and $\Upsilon(t^1) = M_{root}$.

Proof by contradiction: Let t_a be the actual Top-1 and $top_{root} \neq t_a$. Let u be the closest node from root, such that $top_u = t_a$, that means $top_{parent(u)} = t_b \neq t_a$. This is because during the merge operation at $parent(u)$, $m_a \prod_{x \leq i < a} c_i < m_b \prod_{x \leq i < b} c_i$, where ℓ_x is the leftmost leaf of $parent(u)$. Multiplying both the sides of the equation

TABLE II

CALCULATION OF rank-scores (WITH $\alpha = 0.9$) OF TUPLES IN TABLE I:
 $t_1, \{t_2, t_4\}, \{t_3, t_6\}, t_5$

Tuple	Prob	m	c	Υ
t_1	0.30	0.300	0.970	0.300
t_2	0.40	0.400	0.960	0.388
t_3	0.20	0.200	0.980	0.186
t_4	0.50	0.521	0.948	0.475
t_5	0.30	0.300	0.970	0.260
t_6	0.45	0.459	0.954	0.385

with $\prod_{i < x} c_i$, we get $\Upsilon(t_a) < \Upsilon(t_b)$, which is a contradiction to the statement that t_a is the highest rank-scored tuple. Therefore $t^1 (= t_a)$ will always be at the root and $M_{root} = m_a \prod_{1 \leq i < a} c_i = \Upsilon(t_a) = \Upsilon(t^1)$. \square

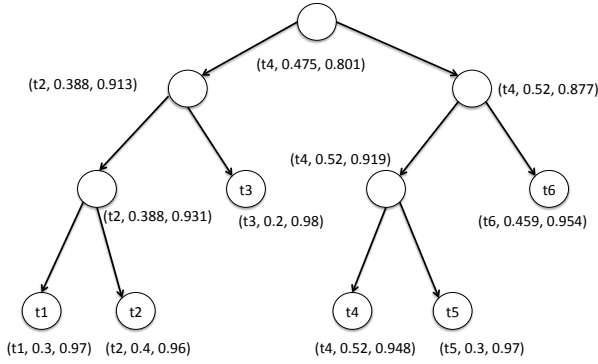


Fig. 1. The data structure for uncertain database in Table II

In the following subsections, we show how to perform different operations such as update-leaf, insert-leaf and delete-leaf on this tree. Later, we use these operations for retrieving Top- k tuples, insertion and deletion of tuples.

A. Update-leaf

The values m_i and c_i within a leaf node ℓ_i can be changed in constant time. But this will change the m and c values at all nodes which are in that path from ℓ_i to root. Therefore we need to perform MERGE operation on all nodes in the path from ℓ_i to root, starting from $parent(\ell_i)$. Since the height of a balanced binary tree is bounded by $O(\log N)$, the total time for update-leaf can also be bounded by $O(\log N)$.

Theorem 4: The m_i and c_i values of a leaf can be updated in $O(\log N)$ time.

B. Insert-leaf and delete-leaf

We first explain, how one-one correspondence between tree leaves and tuples in relation T can be maintained during

insertion or deletion of a leaf.

- **Insert:** To insert a new leaf, we begin by carrying out standard insert procedure of a binary search tree, which would create a new leaf node v . Let w be the parent of this newly created node. Node w being the leaf prior to insertion of v , represents a single tuple from T and should remain as a leaf after insertion of v as well. This can be achieved by creating a new internal node u , which becomes the parent of v and w .
- **Delete:** If deletion of a node results in an internal node with only one child, we perform recursive delete on that internal node.

After insert or delete of a leaf node ℓ_i , we need to update the M and C values at each node along the path of insertion or deletion. This can be achieved by performing MERGE operation in bottom-up fashion beginning with $parent(\ell_i)$. If tree goes out of balance after insert or delete, necessary rebalancing may force further re-computation at nodes whose left or right subtree is changed. However, such nodes are bounded by the height ($O(\log N)$) of the tree. Hence Insert-leaf and leaf-delete operations can be done $O(\log N)$ time.

C. Retrieving Top- k tuples

In theorem 3, we proved that, by MERGE operation the Top-1 tuple t^1 will be the propagated to root node as top_{root} . Therefore t^1 can be retrieved in constant time. In order to retrieve the Top-2 tuple t^2 , we use the following strategy. After retrieving t^1 , we set $\Upsilon(t^1) = 0$. As a result, the next highest rank-scored tuple t^2 will be propagated as top_{root} instead of t^1 . This can be achieved by performing Update-leaf operation on leaf ℓ_j (leaf representing the current $top_{root} = t_j$), with it m_j value set to zero. As c_j remains unchanged, update operation affects only the computation of rank-score of t_j leaving rank-score of all other tuples unchanged. Repeating the same process, we can retrieve top- k tuples with highest rank-score values. We can revert back the changes done in data structure for answering top- k query by restoring the m values for k retrieved tuples using Update-leaf operation.

Top- k

for $i = 1$ to k

$t_j = top_{root}$

report top_{root} as top- i tuple

Update-leaf(t_j) with $m_j = 0$

Figure 2 shows an example for retrieving Top-2 tuple from the uncertain data in table I.

Theorem 5: Top- k rank-scored tuples can be retrieved in $O(k \log N)$ time.

Proof: For every tuple t_j retrieved for answering top- k query, we perform Update-leaf operation twice: once for setting $m_j = 0$ so that tuple with next highest rank-score can be retrieved and next after reporting top- k answers so as to

restore the tree changes. Since `Update-leaf` is a $O(\log N)$ time operation, total time for `Top-k` retrieval can be bounded by $O(k \log N)$.

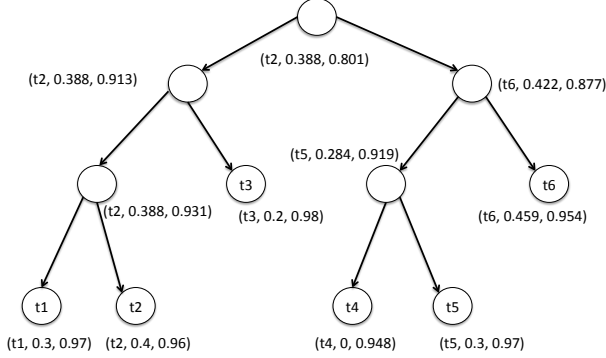


Fig. 2. The data structure after setting $m_4 = 0$ for retrieving Top-2

D. Insert-tuple and delete-tuple

Whenever a tuple t_i gets inserted(deleted) from relation T , we modify our data structure as follows:

- We begin by carrying out `Insert-leaf` or `leaf-delete` operation as necessary. If t_i is an independent tuple then at this point all nodes in the tree Δ have correct values for C and M . Hence no further action is necessary.
- If t_i is not independent, then its insertion(deletion) will change m_j and c_j values for all leaf nodes corresponding to tuple t_j such that $j > i$ and $\tau(t_i) = \tau(t_j)$. These change can be accommodated by performing `Update-leaf` operation on each ℓ_j .

Figure 3 shows an example of inserting a new tuple t^* (with $score(t_2) > score(t^*) > score(t_3)$) and is mutually exclusive with t_5 in the uncertain data in table II and figure 4 shows an example for deletion of a tuple.

Thus insertion(deletion) of a tuple can result in one `Insert-leaf` or `leaf-delete` operation and at max $|\tau(t_i)|$ `Update-leaf` operations. Since any x -tuple can have only constant number of operations, tuple insertion and deletion can be handled in $O(\log N)$ time. We note that updating of tuples can be simulated by first deleting and then reinserting it with updated values.

We summarize the space requirement and performance of the proposed data structure in the following theorem.

Theorem 6: A collection of uncertain data can be maintained using a linear size dynamic data structure, which can retrieve Top- k rank-scored tuples in $O(k \log N)$ time, and can support insertion or deletion of a tuple t in $O(d \log N)$ time, where d is the number of tuples which are related to t . \square

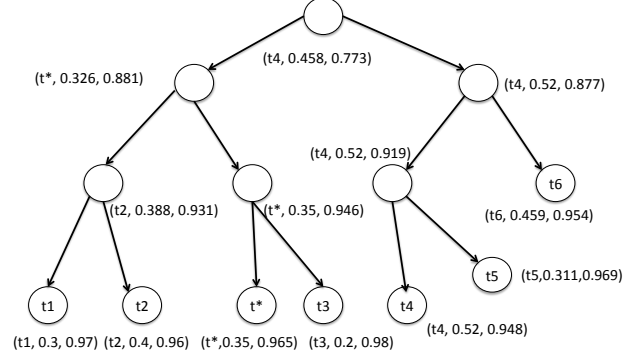


Fig. 3. The data structure in fig1 after inserting t^*

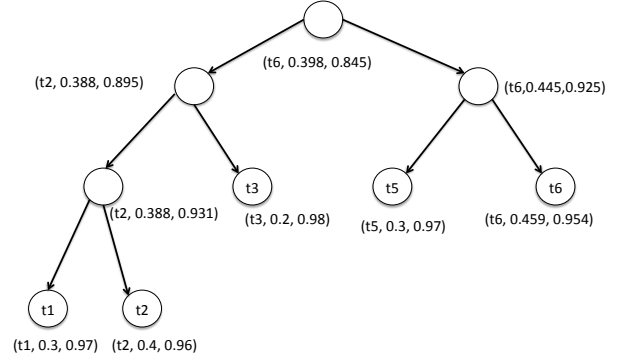


Fig. 4. The data structure in fig1 after deleting t_4

VI. EXPERIMENTAL STUDY

In this section, we present an experimental study with both synthetic and real data evaluating effectiveness of the data structure in handling changes in underlying database and answering top- k queries. All experiments were conducted on 2.4 GHz Intel Core 2 Duo machine with 2GB memory running MAC OS 10.6.4.

Datasets: We created a synthetic dataset containing 1,00,000 tuples. Score of a each tuple is chosen uniformly at random from $[0, 1000000]$ and its probability is uniformly distributed in $(0.5 \times 10^{-5}, 1.5 \times 10^{-5})$. The number of tuples involved in each x -tuple follows the uniform distribution (2,10).

Along with synthetic datasets, we also use International

Ice Patrol(IIP) Iceberg Sighting Database ¹. Each sighting record in the database contains date, location, number of days the iceberg has drifted, etc. As it is crucial to detect the icebergs drifting for long periods, we use the *number of days drifted* as ranking score. The sighting record is also contains a confidence-level attribute according to the source of sighting: R/V (radar and visual), VIS (visual only), RAD (radar only), SAT-LOW (low earth orbit satellite), SAT-MED (medium earth orbit satellite), SAT-HIGH (high earth orbit satellite), and EST (estimated). We converted these seven confidence levels into probabilities 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, and 0.4 respectively. We gathered all records from 1981 to 1991 and 1998 to 2004. Based on it then we created 1,00,000 tuples dataset by repeatedly selecting records randomly.

Results: For all of our experiments we choose $\alpha = 1 - 0.9^{50}$. We begin by evaluating the query performance of the data structure. We retrieve top- k tuples from both the datasets for k ranging from 10 to 100. Linear dependance of query time as obtained in the time bounds is evident from the results show in Figure 5. Also we can note that, correlations among tuples does not affect the query time of our data structure.

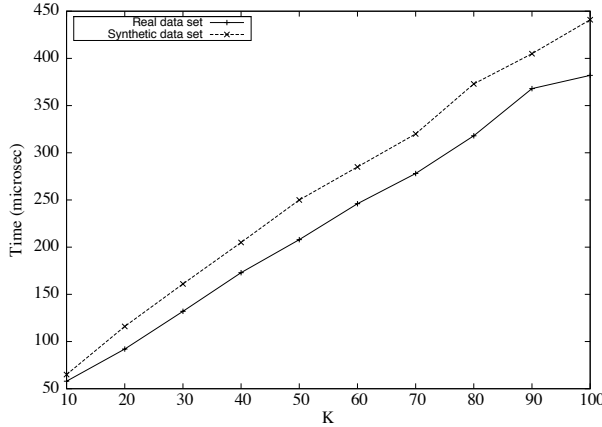


Fig. 5. Top- k query performance on real and synthetic data

Next set of experiments conducted shows efficiency of our data structure in handling tuple insertions and deletions. Time required for inserting and deleting 100 tuples is measured for datasets of varying sizes. Figure 6 and 7 shows that processing time per tuple increase slowly with data size. Whenever a tuple is inserted or deleted, to maintain the correctness of data structure, we also need to update information for leaves corresponding to its related tuples. As all tuples in real data set are assumed to be independent average insertion/deletion time of a tuple is less than in case of synthetic data having correlations. This can be seen from the results in figure 6 and 7. For synthetic dataset, we insert a tuple in dataset such that it is related to existing tuples. We ensure the x -tuple probability

to be less than 1 to which new tuple is inserted. For deletion, victim tuple is selected at random. Figure 6 and 7 also shows the effect of varying data size on query performance of data structure.

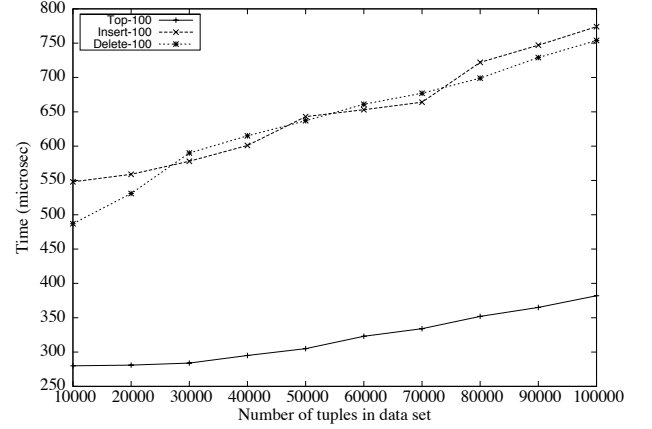


Fig. 6. Processing (insert, delete, top- k) cost on real dataset

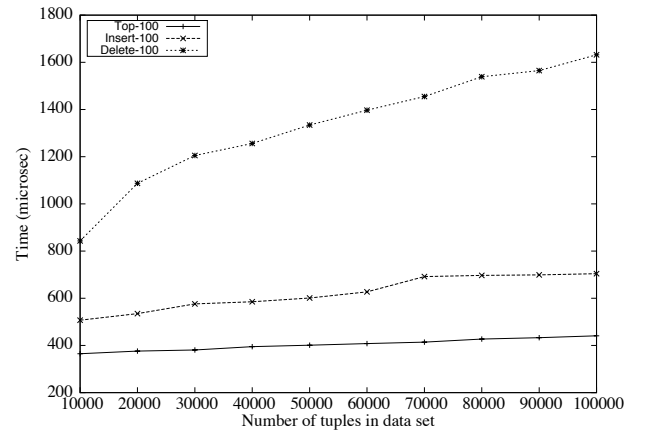


Fig. 7. Processing (insert, delete, top- k) cost on synthetic dataset

Data structure proposed in this paper can be used when data arrives in streaming fashion. Jin et al. [5] have studied the problem of answering top- k queries on sliding windows. Our data structure achieves performance comparable to synopses proposed by them in terms of handling tuple insertion and deletions. Even though our data structure takes linear size as compared to these space efficient synopses, it can be noted that they rely on random order stream model used in streams algorithm community [13], [14], [16] and in worst case would take linear size as well.

¹<http://nsidc.org/data/g00807.html>

VII. RELATED WORK

Uncertain data management has attracted a lot of attention in recent years due to an increase in the number of application domains that naturally generate uncertain data. These include sensor networks [17], data cleaning [18] and data integration [19], [20]. Several probabilistic data models have been proposed to capture data uncertainty (e.g. TRIO [11], MYSTIQ [3], MayBMS [6], ORION [1], PrDB [9]). Virtually all models have adopted possible worlds semantics. Each data model captures tuple uncertainty (existence probabilities are attached to the tuples of the database), or attribute uncertainty (probability distributions are attached to the attributes) or both. Further distinction can be made among these models based on support for correlations. Most of the work in probabilistic databases has either assumed independence or supports restricted correlations, mutual exclusion being the most common. Recently proposed approaches [9], [7] extend the support for any arbitrary correlations.

Efforts have been made in recent times to extend the semantics of “top- k ” to uncertain databases. Soliman et al. [10] defined the problem of ranking over uncertain databases. They proposed two ranking functions, namely U-Top k and U- k Ranks, and proposed algorithms for each of them. Improved algorithms for the same ranking functions were presented later by Yi et al. [12]. Hua et al. [4] proposed another top- k definition PT- k (*probabilistic threshold queries*) and proposed efficient solutions. Cormode et al. [2] defined number of key properties satisfied by “top- k ” over deterministic data including exact- k , containment, unique-rank, value-invariance, and stability. With each of the existing top- k definition lacking one or more of these properties, Cormode et al. [2] proposed yet another ranking function expected-rank. As the list of top- k definitions continued to grow, Li et al. [8] argued that a single specific ranking function may not be appropriate to rank different uncertain databases and empirically illustrated the diverse, conflicting nature of parameterized ranking functions that generalize or can approximate many known ranking functions.

With most of the work for top- k query processing being focused on “one-shot” top- k query for static uncertain data, Chen and Yi [15] was the first to address the dynamic aspect of uncertain data. They proposed a fully dynamic data structure to support arbitrary insertions and deletions. For an uncertain relation with N tuples, the structure of [15] answers top- k queries in $O(k + \log N)$ time, handles an update in $O(k \log k \log N)$ time and takes $O(N)$ space. However, this structure is tied to a single ranking function i.e. U-Top k and works only for independent tuples. Moreover, it can be built for some fixed k value and cannot answer a top- j for $j > k$. Dependence of time, required for handling update, on k is also not desirable. Recently, Jin et al. [5] proposed a framework for sliding window top- k queries on uncertain streams supporting several ranking functions. This framework assumes random-order stream model (tuples arrive

in a random order) which significantly reduces the space requirement as compared to the worst-case scenario in which any data structure will have to remember every tuple in the current window.

VIII. CONCLUSIONS

In this paper we present a dynamic data structure, which can retrieve top- k tuples in $O(k \log N)$ time and has update cost of $O(\log N)$. We also evaluate efficiency of proposed data structure with experiments using synthetic and real data. It is an open question if, we can improve the top- k retrieval time to $O(k + \log N)$ without sacrificing update time or is there any lower bound for this problem?

REFERENCES

- [1] R. Cheng, D. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In SIGMOD, 2003.
- [2] G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data and expected ranks. In ICDE, 2009.
- [3] Nilesh Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. In VLDB, 2004.
- [4] M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: A probabilistic threshold approach. In SIGMOD, 2008.
- [5] C. Jin, K. Yi, L. Chen, J. Xu Yu, and X. Lin. Sliding-window top- k queries on uncertain streams. In VLDB, 2008.
- [6] C. Koch. MayBMS: A System for Managing Large Uncertain and Probabilistic Databases. Chapter in Managing and Mining Uncertain Data, C. Aggarwal ed., Springer, 2009.
- [7] C. Koch and D. Olteanu. Conditioning Probabilistic Databases. In VLDB, pages 313325, 2008.
- [8] J. Li, B. Saha, and A. Deshpande. A unified approach to ranking in probabilistic databases. In PVLDB, pages 502-513, 2009.
- [9] P. Sen, A. Deshpande, and L. Getoor. PrDB: Managing and Exploiting Rich Correlations in Probabilistic Databases. VLDB Journal, 2009.
- [10] M. Soliman, I. Ilyas, and K. C. Chang. Top- k query processing in uncertain databases. In ICDE, 2007.
- [11] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In CIDR, 2005.
- [12] K. Yi, F. Li, D. Srivastava, and G. Kollios. Efficient processing of top- k queries in uncertain databases. In ICDE, 2008.
- [13] A. Chakrabarti, G. Cormode, and A. McGregor. Robust lower bounds for communication and stream computation. In Proc. of STOC, 2008.
- [14] A. Chakrabarti, T. Jayram, and M. Patrascu. Tight lower bounds for selection in randomly ordered streams. In Proc. of SODA, 2008.
- [15] J. Chen and K. Yi. Dynamic structures for top- k queries on uncertain data. In Proc. of ISAAC, 2007.
- [16] S. Guha and A. McGregor. Approximate quantiles and the order of the stream. In Proc. of PODS, 2006.
- [17] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In VLDB, 2004.
- [18] A. Halevy, A. Rajaraman, and J. Ordille. Data integration: the teenage year. In VLDB, 2006.
- [19] H. Galhardas, D. Florescu, and D. Shasha. Declarative data cleaning: Language, model, and algorithms. In VLDB, 2001.
- [20] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In SIGMOD, 2003.